A2: RECURRENCES, INDUCTION, DIVIDE AND CONQUER

"Divide each difficulty into as many parts as feasible and necessary to resolve it." —Rene Descartes

Course: CS 5006
Summer 2018
Due: 25 May 2018, 5pm

OBJECTIVES

After you complete this assignment, you will be comfortable with:

- Mergesort
- Induction
- Solving recurrences with substitution
- Solving recurrences with recursion trees/unrolling

RELEVANT READING

- Kleinberg and Tardos, Chapter 5

AND NOW ON TO THE FUN STUFF.

Problem 1: Induction (8 points)

Use mathematical induction to prove the following statements.

(a) (4 points) Prove $3 + 3*5 + 3*5^2 + \cdots + 3*5^n = 3(5^{n+1}-1)/4$ when n is a non-negative number.

Base Case: $n = 1;\ 3 + 3*5^1 = \frac{3(5^{1+1}-1)}{4};\ 3 + 3*5 = \frac{3*(5^2-1)}{4};\ 3 + 15 = \frac{3*(25-1)}{4};\ 18 = \frac{3*24}{4};\ \ 18 = \frac{72}{4};\ 18 = 18$

Assume the equation works for n.

$$n = (n+1)\ Case; 3 + 3*5 + 3*5^2 + \cdots 3*5^n + 3*5^{n+1} = 3*\frac{5^{n+1+1}}{4};$$

$$3*\frac{5^{n+1}}{4} + 3*5^{n+1} = 3*\frac{5^{n+2}}{4}; \frac{3(5^{n+1}) + 12(5^{n+1})}{4} = 3*\frac{5^{n+1+1}}{4}; \frac{15(5^{n+1})}{4}$$

$$= \frac{3(5^{n+2})}{4}; \frac{3*5*(5^{n+1})}{4} = \frac{3(5^{n+2})}{4}; \frac{3*5^1*(5^{n+1})}{4}$$

$$= \frac{3(5^{n+2})}{4}; \frac{3(5^{n+1+1})}{4} = \frac{3(5^{n+2})}{4}; \frac{3(5^{n+2})}{4} = \frac{3(5^{n+2})}{4};$$

(b) (4 points) Show that 5 divides $n^5 - n$ whenever n is a non-negative integer.

Base Case : $n = 1; \frac{1^5-1}{5}; \frac{1-1}{5}; \frac{0}{5}; 0$

Assume equation true for n

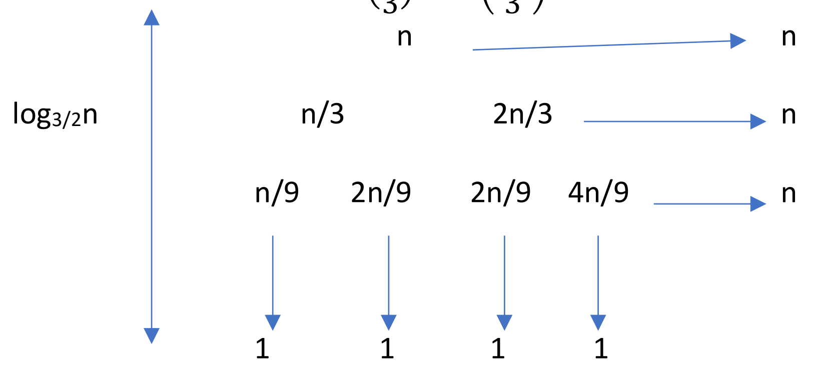
$$n = n + 1\ Case; \frac{(n+1)^5 - (n+1)}{5}; \frac{\left((n^2+2n+1)(n^2+2n+1)(n+1)\right) - (n+1)}{5};$$
$$\frac{\left((n^4+4n^3+6n^2+4n+1)(n+1)\right) - (n+1)}{5};$$
$$\frac{(n^5+5n^4+10n^3+10n^5+5n+1) - (n+1)}{5}; \frac{n^5+5n^4+10n^3+10n^2+4n}{5};$$
$$\left(\frac{1}{5}\right)n^5 + n^4 + 2n^3 + 2n^2 + \left(\frac{4}{5}\right)n$$

Problem 2: Solving Recurrences: Recurrence Trees (8 points)

For each of the following recurrences, draw out the recurrence tree and solve the recurrence.

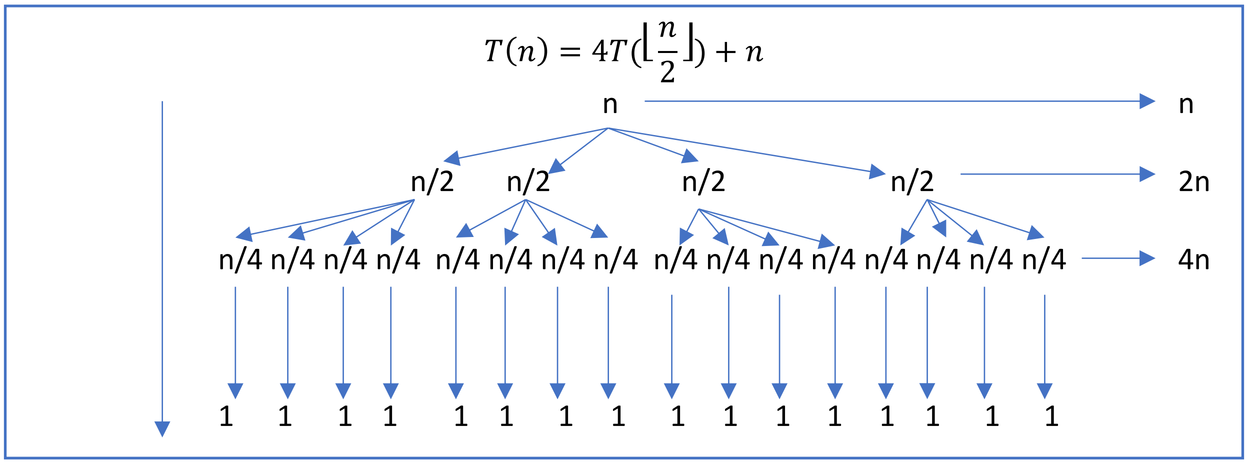
(a) (4 points) Use a recursion tree to show that the solution to the recurrence $T(n) = T(n/3) + T(2n/3) + n$ is $\Omega(n \lg n)$. Don't just draw the tree; use it to explain the answer.

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + n$$



As you split the original problem into smaller problems, the problems on the left side of the equation will be split by $\left(\frac{2}{3}\right)^i$ with "i" being the amount of times you need to split it before you reach 1. So, eventually $n * \left(\frac{2}{3}\right)^i = 1; n = \left(\frac{3}{2}\right)^i; \log_{\frac{3}{2}} n = i$. Since you get n each row, the time of the program could be written as $n \log_{\frac{3}{2}} n + n$ which would be bigger than $n \lg n$ as you get closer to infinity.

(b) (4 points) Use a recursion tree to provide tight bounds for the solution to $T(n) = 4T(\lfloor n/2 \rfloor) + n$.


$T(n) = 4T(\lfloor \frac{n}{2} \rfloor) + n$
n
n/2 n/2 n/2 n/2
n/4 n/4 n/4 n/4 n/4 n/4 n/4 n/4 n/4 n/4 n/4 n/4 n/4 n/4 n/4 n/4
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
n
2n
4n

$n * \left(\frac{1}{2}\right)^i = 1; n = 4^i; \log_4 n = i$. Program runtime $2^i n\ \log_4 n + 2^i n = \Theta 2n \log_4 n + 2n$

Problem 3: Solving Recurrences: Substitution (6 points)

Solving the following recurrences using the substitution method.

(a) (3 points) Show that $T(n) = T\left(\left\lceil \frac{n}{2} \right\rceil\right) + 1$ is $O(\lg n)$.

Original recurrence: $T(n) = T\left(\left\lceil \frac{n}{2} \right\rceil\right) + 1$

Guess: $O(\lg n)$

$T(n) \le c * \lg n ; T\left(\frac{n}{2}\right) \le c \lg\left(\frac{n}{2}\right); T(n) = T\left(\frac{n}{2}\right) + 1 \le c \lg\left(\frac{n}{2}\right) + 1; using\ \log_b\left(\frac{1}{a}\right)$
$= -\log_b a ; so\ T(n) = T\left(\frac{n}{2}\right) + 1 \le c \lg n - c \lg 2 + 1; T(n) = T\left(\frac{n}{2}\right) + 1$
$\le c \lg n - c + 1; T(n) \le c \lg n$

(b) (3 points) Show that $T(n) = 2T(\lfloor n/2 \rfloor) + 17$ is $O(n \lg n)$

Original recurrence: $T(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 17$

Guess: $O(n \lg n)$

$$T(n) \le c * n \lg n ; T\left(\frac{n}{2}\right) \le c\left(\frac{n}{2}\right) \lg\left(\frac{n}{2}\right); T(n) = 2T\left(\frac{n}{2}\right) + 17$$
$$\le 2\left(c\left(\frac{n}{2}\right) \lg\left(\frac{n}{2}\right)\right) + 17;$$
$$\le cn * \lg n * cn \lg\left(\frac{1}{2}\right) + 17;$$
$$using\ \log_b\left(\frac{1}{a}\right) = -\log_b a ; so\ T(n) = 2T\left(\frac{n}{2}\right) + 17 \le cn * \lg n - cn \lg 2 + 17;$$
$$\le cn \lg n - cn + 17; T(n) \le cn \lg n$$

Problem 4: Applications (9 points)

For each of the following problems, outline a reasonable method of solving each. Give the order of the worst-case complexity of your solution.

(a) (3 points) You own a small business. You send out invoices (these days, perhaps digitally, but it might help to think about this as paper). Customers pay these invoices with checks (again, these days, perhaps digitally, but think about it either as a paper check or e.g. an email/notification that you got paid a certain amount by a certain person). Given this stack of invoices and checks, figure out who hasn't yet paid.

```
SoI = invoices; SoC = checks,

        If(SoI > 1 && SoC > 1){
                Apply this to both;
                MergeSort(SoI) based on price charged; n log n
                MergeSort(SoC) based on price charged; n log n
                }
        Else{
                Apply to the one that was bigger than 1
        }
        j =0;
        For (i = 0; i<length(SoI); i++){
                For(k=0; k<length(SoC); k++){
                If(customer and price of SoI[i] != SoC[i]{
                        ListOfPeople[j++] = SoI[i]
```

(b) (3 points) In your small business, you have a list of products that you sell. In this list, each product has a name/description, a manufacturer, an SKU number, and the wholesaler you procure this product from (which may or may not be the same as the manufacturer). You have a list of the 30 wholesalers you purchase from. How many products do you purchase from each wholesaler?

(c) (3 points) You have a stack of invoices to customers that you've collected over the past year. Figure out how many people made at least one purchase from you.

Problem 5: Mergesorting (2 points)

Illustrate the operation of merge sort on the array: [41; 52; 26; 38; 57; 9; 49].

```
[41; 52; 26; 38; 57; 9; 49]
[41; 52; 26; 38] [57; 9; 49]
[41; 52] [26; 38] [57; 9] [49]
[41] [52] [26] [38] [57] [9] [49]

Is 41 less than 52; Is 26 less than 38; Is 57 less than 9; 49 is stable
        Yes                 yes                      no               yes

[41;52] [26;38] [9 – goes in first because it is smaller; 57] [49]
Is 41 < 26; is 9<49?
    No          yes
26 < everything in [41;52] array; 9 < everything in 49 array

[26; 38 – same check done as above results in this getting inserted; 41; 52] [9] is 57 < 49
                                                                                       No


[26; 38; 41; 52] [9; 49 – cause it smaller; 57]

[9; 26; 38; 41; 49; 52; 57] – final result
```

Problem 6: Insertion Sort with Mergesort (15 points)

For this set of problems, we're going to introduce Insertion Sort.

```
InsertionSort(A)
1  for $j = 2$ to A:length
2      $key = A[j]$
3      ▷ Insert A[j] into the sorted sequence A[1 .. j-1 ]
4      $i = j - 1$
5      while $i > 0$ and $A[i] > key$
6          $A[i + 1] = A[i]$
7          $i = i - 1$
8      $A[i + 1] = key$
```

You can think of this like sorting a card hand. Given a stack of cards dealt to you on the table, pick up the top card, put it into your hand in the right place. Take the next card, put it into your hand in the right place. Continue on, at each point, putting the new card in your hand in the right order.

(a) (2 points) What's the worst-case runtime of insertion sort?

The worst-case runtime for insertion sort is $O(n^2)$. Going through the for loop "for j=2 to A:length" is going to take n times long. With the while loop, in the worse-case scenario, you are going to have to go through array every time you add a new element through insertion sort, which will give you a run time of n. So worst case scenario would be $O(n^2)$.

The constant factors in insertion sort can make it faster in practice for small problem sizes on many machines. Thus, it makes sense to use insertion sort within merge sort when subproblems become sufficiently small. Consider a modification to merge sort in which $n = k$ sublists of length k are sorted using insertion sort and then merged using the standard merging mechanism, where k is a value to be determined.

(b) (3 points) Show that insertion sort can sort the $n = k$ sublists, each of length k, in $\Theta(nk)$ worst-case time.

Okay, so the worse-case scenario for normal insertion sort is $O(n^2)$. In the situation where the length of the array is k instead of n, the worse time to run an insertion sort is $O(k^2)$ for one array of k size. In the situation you are taking n elements and splitting it into a number of lists of size k, so the total amount of arrays you are using are $\left(\frac{n}{k}\right)$. Since you are going to do insertions sort on all these arrays, the total amount of time for insertion sort would be $\left(\frac{n}{k}\right) * k^2 = \Theta(nk)$.

(c) (3 points) Show how to merge the sublists in $\Theta(n \lg n/k)$ worst-case time.

Merging lists is generally O(n). Since you'll have $\left(\frac{n}{k}\right)$ amount of problems, when you combine them together it will take n time to do so. Unlike in a situation where you are trying to split n elements still you get n subproblems elements out, you are trying to split n elements into an amount of array with k elements, so while the amount of time it takes to combine them is k, you are only getting $(\frac{n}{k})$ problems by the end, so the time it take to get to this level, would be $\log_{levels\ to\ get\ to\ (\frac{n}{k})}(\frac{n}{k})$. Which would be generalized to $\log(\frac{n}{k})$.

$\mathrm{n} * \log(\frac{n}{k})$ is the total time for the function.

(d) (3 points) Given that the modified algorithm runs in $\Theta(nk + n\lg(n/k))$ worst-case time, what is the largest value of k as a function of n for which the modified algorithm has the same running time as standard merge sort, in terms of Θ-notation?

$$n\lg n = nk + n\lg(\frac{n}{k})\,; n\lg n = n\left(k + \lg\left(\frac{n}{k}\right)\right); \lg n = k + \lg(\frac{n}{k})\,; \lg n = k + \lg n - \lg k\,; 0 = k - \lg k\,; \lg k = k; 2^k = k;$$

(e) (2 points) How should we choose k in practice?

In practice, we should choose a k that will decrease n to a small enough amount that we can take advantage of the speed of insertion sort, while not making it so small that it would be about the same time or longer than if we did a normal merge sort without insertion sort.

(f) (2 points) Suppose we are comparing implementations of insertion sort and merge sort on the same machine. For inputs of size n, insertion sort runs in $8n^2$ steps, while merge sort runs in $64n \lg n$ steps. For which values of n does insertion sort beat merge sort?

$$8n^2 = 64\, n \lg n\, ; 8n = 64 \lg n\, ; n = 8 \lg n\, ;$$

Any values of n that are lower than 8 lg n

Qestion	Points	Score
Induction	8	
Solving Recurrences: Recurrence Trees	8	
Solving Recurrences: Substitution	6	
Applications	9	
Mergesorting	2	
Insertion Sort with Mergesort	15	
Total:	48	

SUBMISSION DETAILS

Things to submit:

- Submit your assignment in your Github repo.
  - The written parts of this assignment as a .pdf named "CS5006_[lastname] _A2.pdf". For example, my file would be named "CS5006_Slaughter_A2pdf". (There should be no brackets around your name).
  - Make sure your name is in the document as well (e.g., written on the top of the first page).
  - Make sure your assignment is in the A2 folder in your Github repo.

HELPFUL HINTS

- Ask clarification questions on Piazza.
- Remember, your write-up should convince graders and instructors that you are providing your own work and should showcase your understanding.
- Use the resources page on the course website for supplemental materials.
- In general, problems will be graded both on whether you are taking the right approach and whether you got the right answer. So, show your work and explain your thinking.